LEON VxWorks-6.5 BSP and SPARC Port

Getting started with LEON BSP under VxWorks version 6.5

Written by Daniel Hellström

VXWORKS-6.5
Version 1.1.1
June 2009

Kungsgatan 12
411 19 Göteborg
Sweden
www.aeroflex.com/gaisler

tel +46 31 7758650
fax +46 31 421407

Table of Contents

1 INTRODUCTION....4
1.1 Content....4
1.2 LEON2/3 peripherals....4
1.3 MMU support....5
1.4 Not supported....5
1.5 Tool chain....5
1.6 VxWorks Tools and Workbench....5
1.6.1 Starting a console under Windows....5
1.7 Debugging VxWorks kernel....5
1.7.1 VxWorks console in GRMON....6
1.8 U-Boot support....6
1.9 Support....6
1.9.1 File SPARC/LEON bug report....6

2 INSTALLATION....8
2.1 Extracting source....8
2.1.1 Installing GNU GCC toolchain....8
2.2 Wind River DIAB C/C++ compiler patch....9
2.3 Copying and Compiling VxWorks....9
2.3.1 Compiling with GCC....9
2.4 Updating VxWorks installation directory....11
2.5 Preparing the Wind River Workbench for SPARC/LEON....12

3 PROJECT CREATION....13
3.1 Using command line tools....13
3.2 Using the Workbench....13

4 PROJECT CONFIGURATION....14
4.1 SPARC/LEON Boot loader configuration....14
4.1.1 BOOTCFG_FREQ_KHZ....15
4.1.2 BOOTCFG_UART_BAUDRATE....15
4.1.3 BOOTCFG_UART_FLOWCTRL....15
4.1.4 BOOTCFG_WASH_MEM....15
4.1.5 BOOTCFG_DSUX_ADDRESS....16
4.1.6 BOOTCFG_MCTRL_MCFG1....16
4.1.7 BOOTCFG_MCTRL_MCFG2....16
4.1.8 BOOTCFG_MCTRL_MCFG3....16
4.1.9 BOOTCFG_SDCTRL_SDCFG....16
4.1.10 BOOTCFG_DDRSPA_CFG1....16
4.1.11 BOOTCFG_DDR2SPA_CFG1....16
4.1.12 BOOTCFG_DDR2SPA_CFG3....16
4.2 Kernel component selection....16
4.3 Kernel component configuration....18
4.3.1 Default kernel boot argument....19
4.4 Kernel memory configuration....19
4.4.1 Base address....19
4.4.2 Memory Size....20

4.5 Driver configuration....20
4.5.1 GRETH, 10/100/1000 Ethernet....20
4.5.2 LAN91C111, 10/100 Ethernet....20
4.5.3 GRSPW, SpaceWire....20
4.5.4 GRCAN, DMA CAN 2.0b....20
4.5.5 OCCAN, non-DMA CAN 2.0b....21
4.5.6 GRUSBHC, Host USB 1.1, 2.0....21
4.5.7 I2CMST, I2C Master....21
4.5.8 APBUART....21
4.5.9 PCI....22
4.5.10 1553....23

5 BUILDING PROJECT....24
5.1 Build rule....24
5.2 Building the project....24
5.3 Running image on hardware using GRMON....25

6 BOOTROM....26
6.1 Creating a bootrom project....26
6.2 Configuring the bootrom....26
6.3 Building the bootrom....27
6.4 Installing and running the bootrom....27

7 RTP PROJECTS USING COMMAND LINE TOOLS....29
7.1 Overview....29
7.2 Creating and Compiling a RTP Project....29
7.3 Inserting an RTP project it into a ROMFS....29
7.4 Building image project with ROMFS and RTPs....30
7.5 Running real time processes on target hardware....30

8 SUPPORT....32

1 INTRODUCTION

This document is intended as an aid in starting developing with LEON based platforms using the LEON VxWorks-6.5 BSP. It briefly takes the reader through some of the most important steps in the software development process such as installing the BSP, creating, configuring and building projects using the LEON BSP. The reader is assumed to have knowledge in VxWorks-6.5.

1.1 Content

This CD contains two board support packages (BSP) for LEON2 and LEON3 processors for use with the Wind River VxWorks-6.5 operating system. The BSPs are designed for use with the Gaisler Research SPARC port of VxWorks-6.5, which also is included in full source. One of the BSPs supports Memory Management Unit (MMU) and the other BSP is intended for use with non-MMU systems. Drivers are common to both BSPs.

The BSP includes drivers for the following functions:

- LEON2 and LEON3
- MMU and non-MMU systems
- FPU, non-FPU, hardware MUL/DIV and software MUL/DIV support
- Interrupt controller
- APBUART UART console/terminal driver
- Timer unit
- 10/100 Ethernet networking (GRETH, LAN91C111)
- 10/100/1000 Ethernet networking (GRETH)
- SpaceWire (GRSPW)
- DMA CAN 2.0 (GRCAN)
- non-DMA CAN 2.0 (OCCAN)
- Host USB 1.1 and 2.0 (GRUSBHC)
- I2C Master (I2CMST)
- PCI support (GRPCI, PCIF)
- PCI support (AT697 PCI for non-MMU BSP)
- 1553 BC, RT and MT support (B1553BRM)
- GR-RASTA-IO PCI Target
- GR-701 PCI Target
- GR-RASTA-ADCDAC PCI Target
- General Purpose I/O (GRGPIO)
- ADC/DAC controller (GRADCDAC)

The BSP is intended to be used with Linux or Windows, with or without Wind River Graphical Workbench.

The final executable will run on both LEON2 and LEON3/GRLIB processors without requiring any specific configuration.

See section below of features not supported.

1.2 LEON2/3 peripherals

The LEON2 on-chip peripherals are assumed to reside in the default address as defined in the

LEON2 VHDL model. Also the interrupt assignment of UART and Timer are assumed to retain their default values. For LEON3 systems, the address and interrupt is read from the on-chip plug&play information, and is assigned during run-time.

1.3 MMU support

One of the two BSPs supports Memory Management Unit (MMU) which offers memory protection between processes and kernel. Full protection can be achieved by not allowing processes to write each other's private memory and off course not to the kernel area. If such a violation occurs the kernel halts the faulting process and the developer can find out what process and what instruction went wrong.

1.4 Not supported

Below is a list of features currently not supported by the SPARC port and/or the BSPs.

- Workbench scope tools
- Hardware watch points and hardware break points in Workbench, however available when using GRMON or TSIM

1.5 Tool chain

Both the Wind River tool chain (DIAB compiler) and the GNU tool chain (GCC compiler) are supported by Gaisler SPARC LEON VxWorks port.

1.6 VxWorks Tools and Workbench

Command line tools are provided with VxWorks. The command line tools introduce an alternative way instead of using the Workbench. For instance, project settings can be viewed and edited from the command line. The Workbench calls a selection of these commands for us in response to GUI actions, for example after pressing the build button. The command line tools have an advantage over the Workbench when it comes to automatizing certain tasks in the development process.

The provided top level Makefile serves both as a convenience for developers as well as a source of examples on how to use the command line tools.

1.6.1 Starting a console under Windows

To access the command line tools in Windows, start with:

```
Start->Run: C:\WindRiver\wrenv.exe -p vxworks-6.5 sh
Press OK
```

Where *C:\WindRiver* is the installation directory of Wind River Workbench.

1.7 Debugging VxWorks kernel

The VxWorks kernel can be debugged using GNU GDB together with the hardware debugger GRMON. GDB can be connected to GRMON using a socket over a network or within the local computer. For instructions on how to use GDB with GRMON see the GRMON manual.

GRMON supports the GDB thread interface for VxWorks making it possible to switch between threads and look at their stacks. One can list threads in GRMON by issuing 'thread info' and list a thread's stack by typing 'thread bt THREAD_ID', similar to GDB 'info thread' and 'thread NUM' followed by 'bt'.

A separate package containing GDB for Linux and Windows is distributed via the FTP service available at ftp://ftp.gaisler.com/gaisler.com/vxworks, vxworks-gdb-6.8-1.0.0-linux.tar.bz2 include precompiled Linux binaries and vxworks-gdb-6.8-1.0.0-mingw.zip include Windows executables. Follow the installation instructions in the README file once the compressed file has been extracted.

1.7.1 VxWorks console in GRMON

The VxWorks console can be accessed from GRMON when starting GRMON with the -u flag, see GRMON manual. In order for GRMON to handle VxWorks backspace correctly "Control+A B" must be typed once the console appears, otherwise VxWorks will echo ? instead of performing the deletion.

1.8 U-Boot support

SPARC/LEON3 U-Boot supports booting VxWorks 6.5 release 1.1.1 and later. The U-boot can be used to download kernel images over network booting directly or updating FLASH. Some of the features are similar to the VxWorks boot rom. However the foot print can be configured much smaller and the FLASH can be managed through a command line interface.

The sources and the SPARC/LEON U-Boot manual can be found on the public Gaisler FTP ftp://ftp.gaisler.com/gaisler.com/u-boot/u-boot-sparc-vxworks-1.0.0.tar.gz and the manual at u-boot-sparc-1.0.0.pdf.

The sources are compiled under Linux using the sparc-elf GNU GCC toolchain freely available at www.gaisler.com.

1.9 Support

For support, contact the Gaisler Research support team at support@gaisler.com.

1.9.1 File SPARC/LEON bug report

For the Gaisler team to service you as fast as possible please provide as much relevant information as possible, see list below for an example

- A description of what steps involved to reproduce the bug
- Compiler used, GCC or DIAB
- BSP used (non-MMU or MMU BSP)
- On what hardware the bug occurs (LEON2/3, MMU, FPU, Board)
- Console output where applicable
- Start up parameters to GRMON or TSIM
- Instruction trace (inst 100), back trace (bt) and register values (reg) of trapped CPU
- The VxWorks image (default/VxWorks) zipped (if possible)

- The project/BSP configuration parameters (prjParams.h) where relevant

2 INSTALLATION

The software package contains not just a BSP, but a complete VxWorks SPARC port. This means that the VxWorks kernel needs to be compiled for the SPARC architecture. During project development the selected BSP is recompiled and linked against the pre built SPARC kernel every time the project is recompiled.

To be able to complete the installation, the following is needed:

- a working VxWorks-6.5 installation
- dist-<relnum>.tar.gz or dist-<relnum>.zip
- GNU tool chain sparc-wrs-vxworks-3.4.4-1.x.y (optional)

For installing VxWorks-6.5 consult your VxWorks documentation.

Windows users start their command sequences by first starting *wrenv* as described in the "VxWorks Tools and Workbench" section.

2.1 Extracting source

Start the installation process by extracting the content of the dist-<relnum>.tar.gz.zip into an empty folder. The ZIP-file is encrypted, please contact support@gaisler.com for a password. This directory will from now on be called *distdir*. Command examples start in the *distdir* directory unless otherwise stated.

Using Linux:

```
$ mkdir /usr/local/gaisler
$ cd /usr/local/gaisler
$ tar -zxvf /path/to/cdrom/dist-<relnum>.tar.gz
```

Using Windows:

```
$ cd C:/vxworks-sparc/
$ unzip dist-<relnum>.tar.gz.zip
$ gunzip dist-<relnum>.tar.gz
$ tar -xf dist-<relnum>.tar
$ cd dist-<relnum>
```

2.1.1 Installing GNU GCC toolchain

SPARC/LEON VxWorks can be used together with GNU GCC or DIAB. DIAB comes with the WindRiver installation DVD whereas the GNU GCC toolchain is available together with the SPARC/LEON VxWorks distribution or can be downloaded from the protected web download area at *www.gaisler.com*. The toolchain is typically named sparc-wrs-vxworks-3.4.4-1.x.y-mingw.zip for Windows hosts and sparc-wrs-vxworks-3.4.4-1.x.y-linux.tar.bz2 for Linux hosts.

The GNU GCC toolchain must be installed under /opt (C:\opt by extracting the zipped toolchain there creating the directory sparc-wrs-vxworks. The compiler binaries in /opt/sparc-wrs-vxworks/bin (C:\opt\sparc-wrs-vxworks\bin) must be added to the search PATH environment variable in order to be found.

2.2 Wind River DIAB C/C++ compiler patch

The Wind River DIAB 5.5.1.0 compiler must be patched in order to avoid a floating point bug introduced. The bug is caused because DIAB does not add all necessary *nop* instructions in between some of the floating point instructions. The patch is available in SPARC/LEON the distribution at *release/WIND00124507fix-6.5[-linux].zip*. The patch must be extracted to *diab/5.5.1.0/* relative to the Wind River VxWorks installation directory.

2.3 Copying and Compiling VxWorks

The distribution contains a script that copies parts of the VxWorks installation into the newly created directory in the previous step. The script needs to know where VxWorks Workbench is installed. The location of the VxWorks directory is stored in the variable VXWORKSDIRCD_PATH. Set the VXWORKSDIRCD_PATH variable at the top of the script file by editing it with a text editor. The script file is located in *distdir* and named Makefile.

During copying, some VxWorks files are overwritten by the patched files that this distribution contains.

The last step in the installation procedure is to compile the kernel, its drivers and libraries that come with the kernel. The command sequence is shown hereafter.

Using Linux:

```
$ cd /path/to/distdir                [distdir]
$ make dist-install                  [copy]
$ make vxworks-compile               [compile VxWorks]
$ make vxworks-lib-compile           [compile VxWorks libs]
```

Using Windows:

```
$ cd C:/vxworks-sparc/dist-<relnum>
$ make dist-install
$ make vxworks-compile
$ make vxworks-lib-compile
```

2.3.1 Compiling with GCC

Create the kernel by adding the following macro definitions to the make command line:

```
$GNU_V8=[y|n] GNU_SOFT=[y|n] make vxworks-compile-gnu
```

and

```
$make vxworks-lib-compile-gnu
```

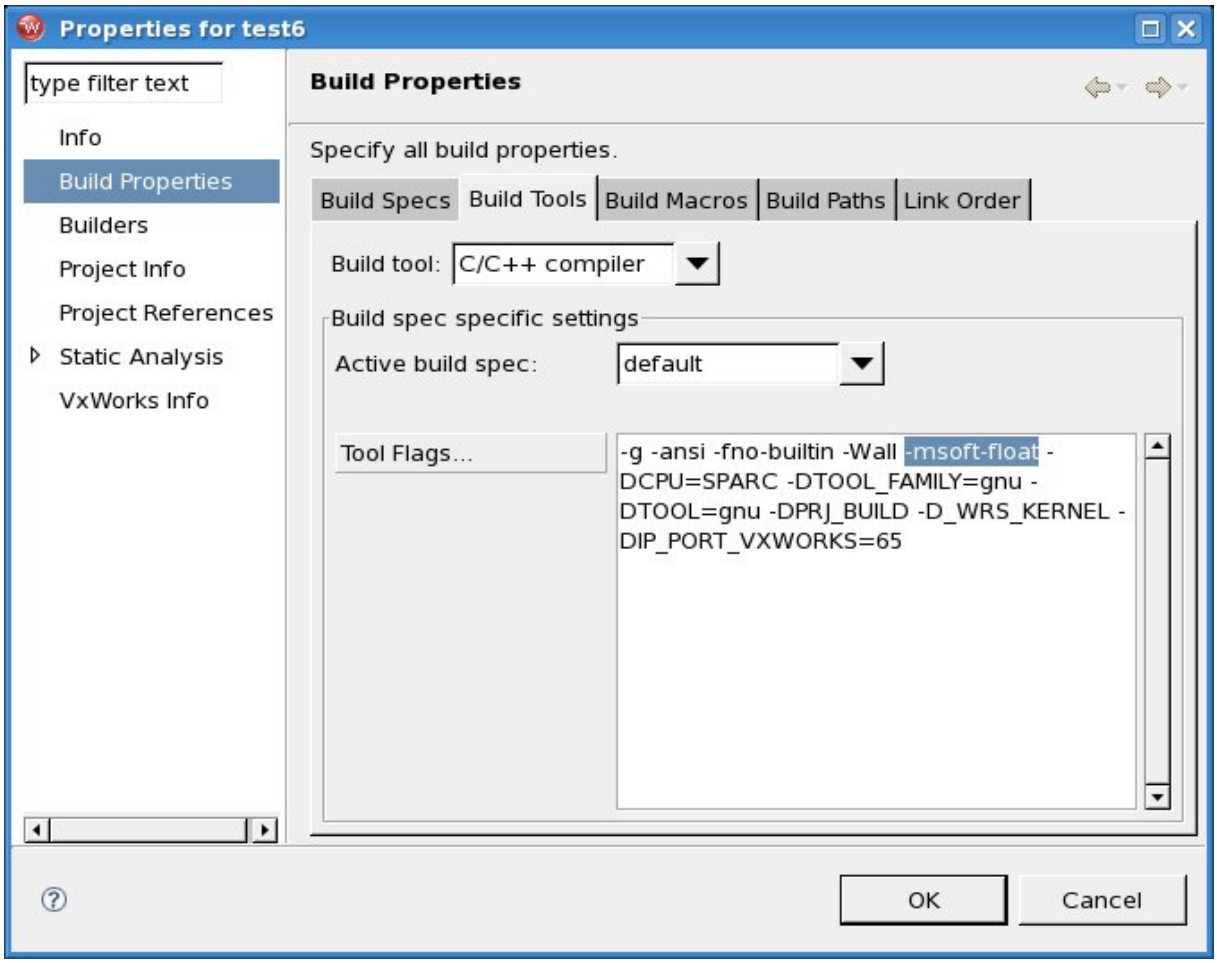
Depending on the value of GNU_V8 (GCC switch -mv8) and GNU_SOFT (GCC switch -msoft-float) different kernels will be built. When recompiling the kernel with a different [GNU_V8, GNU_SOFT] combination the make rule

```
$make vxworks-cleaner
$make vxworks-lib-clean
```

should be issued first. Target vxworks-compile-gnu will create files

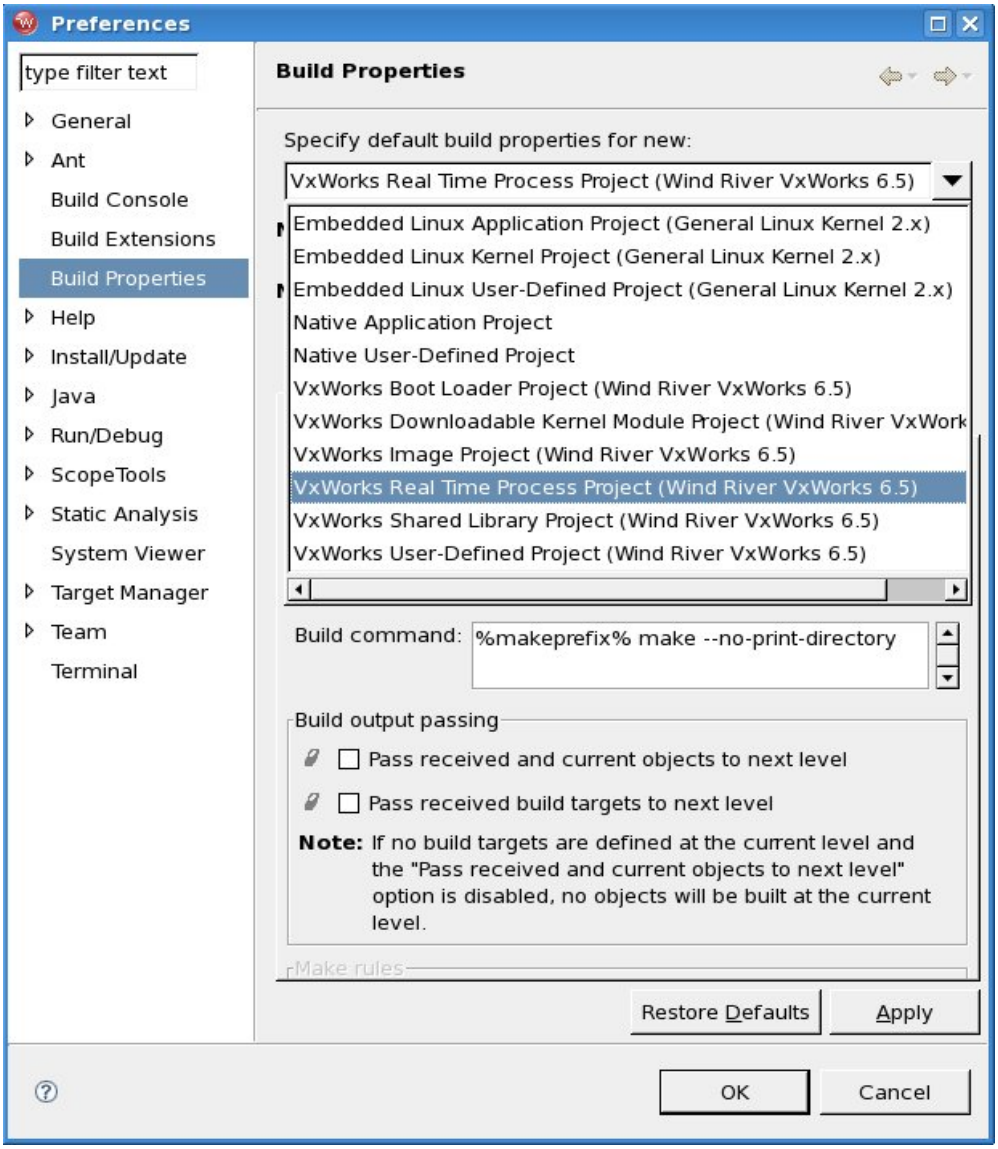
- $(WIND_USR)/tool/gnu/opt.SPARCgnu
- $(TGT_DIR)/h/tool/gnu/opt.SPARCgnu

that are used to append the appropriate GCC switches to the GCC command.



When you change the [GNU_V8, GNU_SOFT] combination you also have to take care to adjust the GNU toolchain flags setting for your already existing Image Project in your workbench. The flags settings is calculated from the [GNU_V8, GNU_SOFT] combination when the image project was created. It has to be changed by hand to match the new [GNU_V8, GNU_SOFT] combination. Right click on the project and select "Properties". Then in the "Build Properties" panel adjust all flags to reflect the new setup.

When you use the workbench to create a RPT application using the GNU toolchain you should reimport the BUILD specifications each time you change the [GNU_V8, GNU_SOFT] combination:


Preferences
type filter text
General
Ant
Build Console
Build Extensions
Build Properties
Help
Install/Update
Java
Run/Debug
ScopeTools
Static Analysis
System Viewer
Target Manager
Team
Terminal
Build Properties
Specify default build properties for new:
VxWorks Real Time Process Project (Wind River VxWorks 6.5)
Embedded Linux Application Project (General Linux Kernel 2.x)
Embedded Linux Kernel Project (General Linux Kernel 2.x)
Embedded Linux User-Defined Project (General Linux Kernel 2.x)
Native Application Project
Native User-Defined Project
VxWorks Boot Loader Project (Wind River VxWorks 6.5)
VxWorks Downloadable Kernel Module Project (Wind River VxWork
VxWorks Image Project (Wind River VxWorks 6.5)
VxWorks Real Time Process Project (Wind River VxWorks 6.5)
VxWorks Shared Library Project (Wind River VxWorks 6.5)
VxWorks User-Defined Project (Wind River VxWorks 6.5)
Build command:
%makeprefix% make --no-print-directory
Build output passing
Pass received and current objects to next level
Pass received build targets to next level
Note: If no build targets are defined at the current level and the "Pass received and current objects to next level" option is disabled, no objects will be built at the current level.
Make rules
Restore Defaults
Apply
OK
Cancel

Go to the Preferences window, under “Build Properties” select “VxWorks Real Time Process Project” and then click on “Restore Defaults”.

2.4 Updating VxWorks installation directory

In order for the Workbench to access the SPARC kernel and the new LEON BSPs the *vxworks-6.5* directory in the Wind River installation directory must be updated containing the new libraries and sources.

For Linux users the far most easiest approach is to change the name of the original *vxworks-6.5* directory in the VxWorks installation directory, and make a symbolic link to the *vxworks-6.5* directory in *distdir*:

```
$ cd /opt/WindRiver
$ mv vxworks-6.5 vxworgs-6.5.ori                    [rename]
$ ln -s /path/to/distdir/vxworks-6.5 vxworks-6.5    [link]
```

Windows lacks support for symbolic links. Instead, Windows users should copy the *vxworks-*

6.5 directory from *distdir* to the Wind River installation directory. But first, one must make sure to rename the original *vxworks-6.5* directory in the VxWorks installation directory, as was the case with Linux.

2.5 Preparing the Wind River Workbench for SPARC/LEON

In order for the Workbench to successfully connect to a SPARC target one must have a dll-file aimed for SPARC development and SPARC configuration files. The SPARC plugins is missing in the Windows release of Wind River Workbench while the plugins for Linux and Solaris is included in the distribution of the Workbench.

The SPARC plugins and the configuration files are distributed in a ZIP-file, release/sparccpuplugin-6.5.tar.gz, found in the Gaisler VxWorks SPARC distribution. Extracting the ZIP-file into the Workbench directory installs the files needed correctly. Below is an example of how to extract the file using the Workbench command shell:

Using Linux:

```
$ cd /opt/WindRiver
$ tar -zxf /path/to/distdir/release/sparccpuplugin-6.5.tar.gz
```

Using Windows:

```
$ gunzip C:/vxworks-sparc/dist-
<relnum>/release/sparccpuplugin-6.5.tar.gz
$ cd C:/WindRiver
$ tar -xf  C:/vxworks-sparc/dist-
<relnum>/release/sparccpuplugin-6.5.tar
```

3 PROJECT CREATION

Projects can be created in two ways; either with the Workbench or with the command line tools that come with VxWorks. For convenience some of the often typed command sequences are prepared in the Makefile file, ready to be used.

At project creation one must specify what BSP the new project should be based upon. There are two BSPs available. The wrleon23_sparcleon BSP is made for designs not using an MMU, while wrleon23mmu_sparcleon is intended for use with an MMU.

All board support packages reside in vxworks-6.5/target/config, and each BSP has its own directory.

3.1 Using command line tools

First start a console. Windows users start a console as described previously:

Start->Run: C:\WindRiver\wrenv.exe -p vxworks-6.5 sh [OK]

One can simply create a new project by typing (in the distdir directory):

```
$ NAME=prj_name make vxworks-usr-create
```

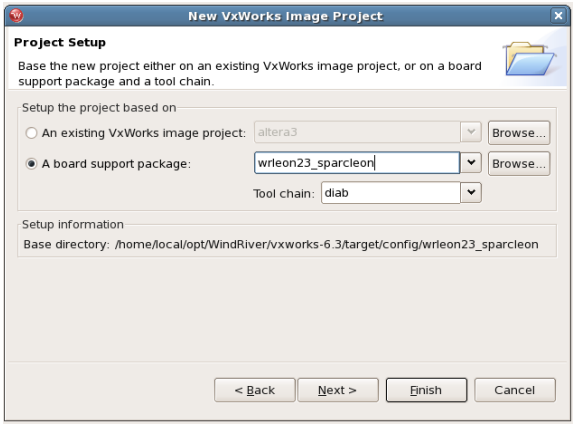
or for projects using MMU

```
$ MMU=y NAME=prj_name make vxworks-usr-create
```

The make script creates a new project based on the MMU-less BSP or the BSP with MMU support by searching the command line for MMU=y. The parameter prj_name is the new project's name. The created project can now be found under usr/prj_name.

3.2 Using the Workbench

A project can be created with Wind River Workbench by selecting “file/new/VxWorks Image Project”. Enter a name for the project, press Next. Select a Board Support Package the project should be based upon, for non-MMU projects choose wrleon23_sparcleon, for systems with MMU where the MMU should be used select wrleon23mmu_sparcleon. When selecting compiler DIAB or GNU GCC it is important to select the same compiler as was used during kernel building in the installation step (vxworks-compile = DIAB, vxworks-compile-gnu = GCC). Then press Finish.



4 PROJECT CONFIGURATION

This section describes the configuration options of the LEON VxWorks 6.5 BSP. Beginning with the most basic configuration, which is the boot loader configuration. Followed by the kernel configuration, selecting components and modifying their parameters. The GRETH Ethernet driver is used as an example on how to include and configure a driver that comes with the BSP.

During project creation defaults are read from the config.h file located in the BSP directory. The defaults can be overridden by using the GUI or the command line tools. If one is certain of using the same hardware for all projects it may be convenient to change the BSP defaults in the config.h file manually with a text editor. Another, and perhaps preferable, way is to create a reference project for coming projects to be based upon.

4.1 SPARC/LEON Boot loader configuration

Depending on what target is being built it may be relevant to configure the boot loader settings. More information about build targets can be found in section building project.

When running the kernel from RAM using GRMON, it is not necessary to configure the boot loader since it is not used. GRMON itself does the basic initialization when typing *run*.

The boot loader's primary task is to load the VxWorks kernel. Some basic initialization such as SDRAM parameters and system console baud rate is initialized by the boot loader before booting VxWorks. The initialization parameters are configured using the VxWorks Kernel configuration utility from the Workbench, see figure 4.1, or by creating a custom *bootcfg.h* file. The default boot loader parameters are set in *bootcfg_def.h*, it can be found in *target/config/wrleon23[mmu]_sparcleon* (BSP directory), they are included into the BSP from *config.h* and can be overridden by the project's configuration as it is included after *bootcfg_def.h*. The boot loader in romInit.s includes the configuration and behave differently depending on the settings.

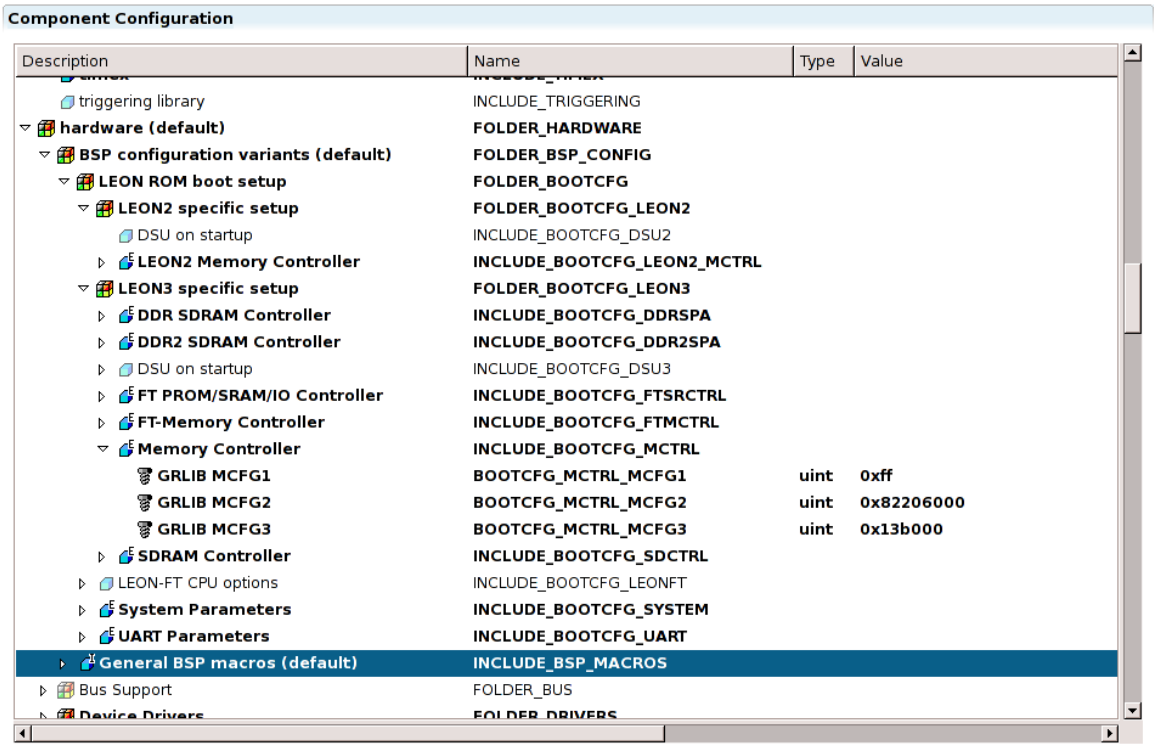


Illustration 4.1: Boot loader configuration

As mentioned above the boot loader settings can be set by creating a custom *bootcfg.h* file or editing *bootcfg_def.h* directly. This changes the defaults of the BSP affecting every newly created project based on that BSP, however it does not affect already created BSPs. However, configuring the boot loader using the Workbench only changes that particular project's boot loader settings.

To find the correct initialization values please consult the core documentation (GR-IP documentation).

Some of the boot loader parameters are briefly described below.

4.1.1 BOOTCFG_FREQ_KHZ

Frequency of processor in KHz. This setting affects the system timer interrupt frequency, baud rates calculated from the system frequency etc.

4.1.2 BOOTCFG_UART_BAUDRATE

The UART baud rate it is first initialized to before entering the VxWorks kernel. The default is 38400 bits/second.

4.1.3 BOOTCFG_UART_FLOWCTRL

Enable or disable UART flow control on start up.

4.1.4 BOOTCFG_WASH_MEM

Enable or disable main memory clearing. All main memory available to VxWorks is washed by setting the contents to zero. Enabling washing can be necessary for fault tolerant systems,

this is because uninitialized data in the main memory have incorrect check bits and will cause a trap when accessed.

4.1.5 BOOTCFG_DSUX_ADDRESS

Debugging the application starting from PROM without GRMON can be hard sometimes, using "grmon -ni" to connect into a target whose application has crashed may reveal much information and even more can be extracted if the DSU trace buffers are enabled first thing during boot. The DSU is early and is made not to rely on the plug and play information requiring the user to set BOOTCFG_DSUX_ADDRESS to the base address of the DSU hardware registers. The default is 0x90000000.

4.1.6 BOOTCFG_MCTRL_MCFG1

MCTRL Memory Configuration Register 1. Configures I/O and PROM accesses. See memory controller documentation for more information on what individual bits to set.

4.1.7 BOOTCFG_MCTRL_MCFG2

MCTRL Memory Configuration register 2. Configures SRAM and SDRAM parameters. See memory controller documentation for more information on what individual bits to set.

4.1.8 BOOTCFG_MCTRL_MCFG3

MCTRL Memory Configuration Register 3. SDRAM refresh rate value.

4.1.9 BOOTCFG_SDCTRL_SDCFG

GRLIB standalone SDRAM controller, SDCTL configuration register.

4.1.10 BOOTCFG_DDRSPA_CFG1

GRLIB standalone 16/32/64-bit DDR266 SDRAM controller control register 1.

4.1.11 BOOTCFG_DDR2SPA_CFG1

GRLIB standalone 16/32/64-bit DDR2 SDRAM controller control register 1.

4.1.12 BOOTCFG_DDR2SPA_CFG3

GRLIB standalone 16/32/64-bit DDR2 SDRAM controller control register 3.

4.2 Kernel component selection

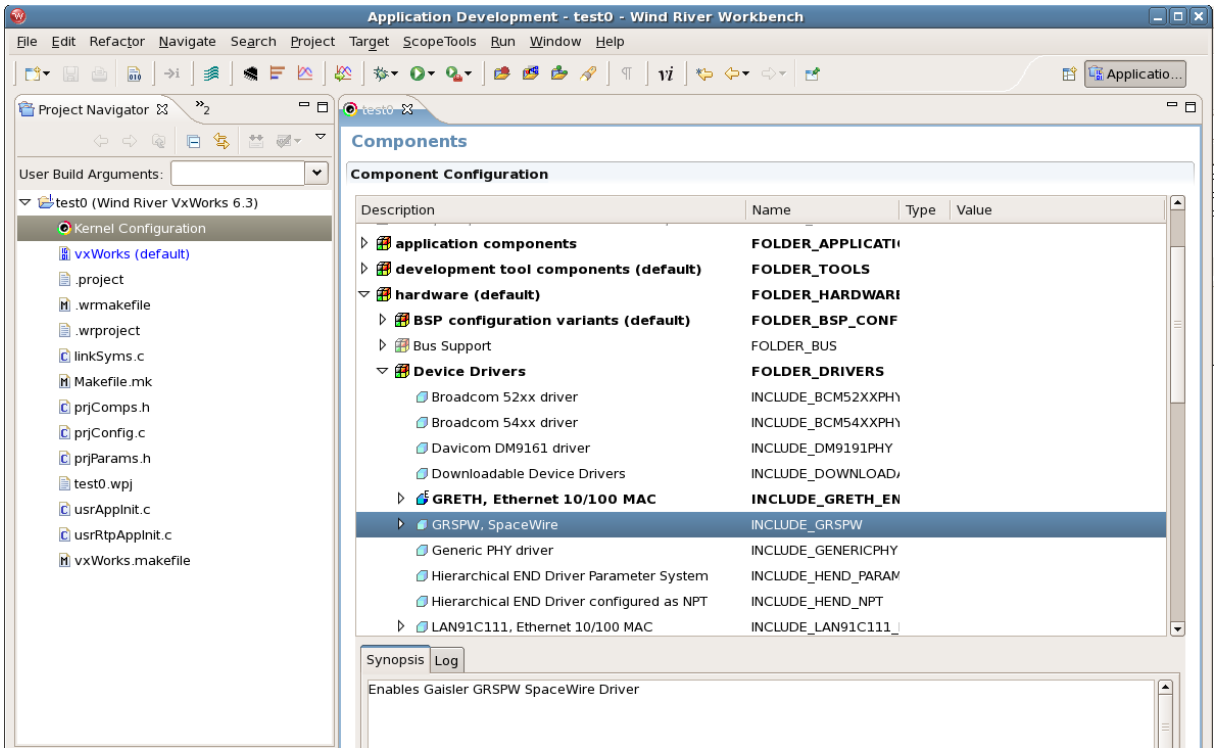
Including components into your project is done as usual using the Workbench or the command line tools.

An example adding and removing the Gaisler Research GRSPW SpaceWire driver component into a test project called *test2* is show below. It may be done with the command line tools:

```
> cd /path/to/distdir
> make execute
$ vxprj component add usr/test2/test2.wpj INCLUDE_GRSPW
adding INCLUDE_GRSPW
$ vxprj component remove usr/test2/test2.wpj INCLUDE_GRSPW
removing INCLUDE_GRSPW
$ exit
>
```

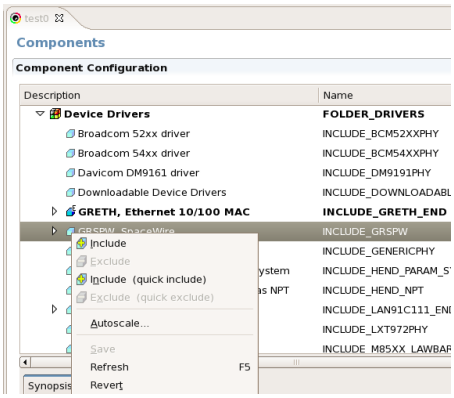
Enter 'vxprj component' in the shell to list possible options for components.

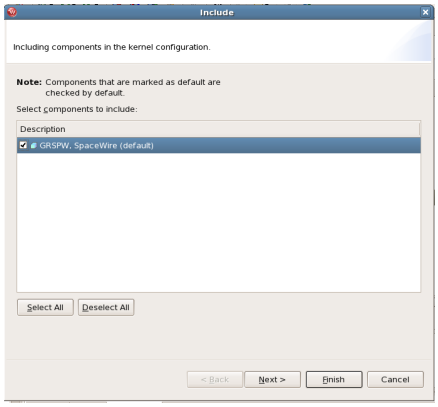
The exact same effect can be achieved using the Workbench. Double click the project's "Kernel Configration" icon in the Workbench to make the kernel configurator appear:



Under "Hardware" BSP specific settings can be found. From "Hardware/Device Drivers" it is possible to select what drivers should be included in the kernel. Right click at a driver and press "Include <DRIVER>" from the pop-up menu. Include only drivers for which hardware is present. Note that not all combinations of drivers may be supported.

Press Finish to include the component or Next to view it's dependencies.





Software drivers can be excluded from the project to save space. Default for a GR-XC3S board is to include the Ethernet driver, but it is also possible to exclude it by right clicking the driver and pressing Exclude.

Generally, when a driver is included the project's prjConfig.c file is updated to call the driver's initialization routine during start up. The driver's options are included into the prjConfig.c file from the prjParams.h file and passed via the initialization routine.

Using an END Ethernet driver such as the GRETH or LAN91C111 driver enables one to take advantage of some of the debugging tools that come with the Workbench. The kernel responds to the user's actions carried out from the Workbench, such as setting breakpoints, pausing and resuming threads. For this to work “WDB END driver connection” must be enabled and the network interface must be configured with a valid IP address. “WDB END driver connection” is enabled by default, but it might be a good idea to review the network interface settings. See the default boot line argument in the next section for more information.

4.3 Kernel component configuration

Each software driver has its own configuration options. The Parameters of a component can be found by expanding a component in the Workbench.

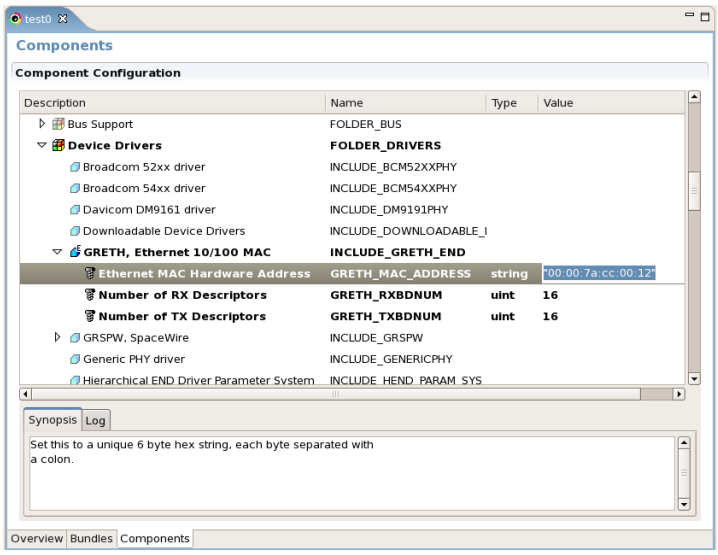
Example on how to use the command line tools to view and change the value of the parameter: LOCAL_MEM_LOCAL_ADRS:

```
> cd /path/to/distdir
> make execute
$ vxprj parameter value usr/test2/test2.wpj \
  LOCAL_MEM_LOCAL_ADRS

LOCAL_MEM_LOCAL_ADRS = 0x40000000
$ vxprj parameter set usr/test2/test2.wpj \
  LOCAL_MEM_LOCAL_ADRS 0x20000000

$ exit
>
```

Following is an example on how to configure the GRETH Ethernet software driver.



To be able to use the Ethernet interface one must configure the Ethernet hardware address of the MAC, this is done by providing input to the MAC's driver. Expand “Hardware”, “Device Drivers” and finally the GRETH driver. All parameters visible to the Workbench are displayed. Double click the value field of the parameter in order to change its value.

RXBDNUM and TXBDNUM defines the number of descriptors(buffers) to be used in the receiving and transmitting parts to the driver. Each descriptor adds about 3k of RAM usage to the driver. The GRETH MAC supports a minimum of one descriptor and a maximum of 128 descriptors in both directions.

4.3.1 Default kernel boot argument

Ethernet interfaces can be set up using the DEFAULT_BOOT_LINE. It can be changed through “Hardware/BSP Configuration variants/General BSP Macros” from the Workbench kernel configuration. IP address and name of host are important parameters to be filled in.

Each Ethernet driver has its own unique name. The name can be used to identify the network driver. The GRETH driver is named “greth” and the LAN Chip driver “lnc”. A certain MAC interface is associated with a unit number, usually it is zero since only one device is present.

See VxWorks documentation on how to set up the boot line argument properly.

4.4 Kernel memory configuration

The LEON VxWorks BSP is not designed for a specific board, but may be used in many different board set ups. Therefore it is often needed to change the default BSP configuration parameters. Memory size and location for instance, may vary between boards.

The BSP memory configuration is set up as any other VxWorks-6.5 project via the Workbench “hardware/memory/BSP Memory Configuration” or using the command line tools as previously described.

4.4.1 Base address

The VxWorks kernel's view of the main memory can be controlled with the parameter

LOCAL_MEM_LOCAL_ADRS. SDRAM memory is usually mapped to 0x40000000.

4.4.2 Memory Size

The size of the board's main memory is set by LOCAL_MEM_SIZE.

4.5 Driver configuration

This section describes driver configuration parameters available from the standard VxWorks kernel configuration utility.

The driver API are documented in a separate document, where available.

4.5.1 GRETH, 10/100/1000 Ethernet

The GRETH Ethernet driver can be found under "Devices Drivers" in the VxWorks kernel configuration utility.

The driver has three configuration parameters, hardware MAC address, number of receive descriptors and the number of transmit descriptors. The MAC address must be unique on the local net. The number of receive descriptors has an impact on performance and memory usage. Each descriptor can hold one packet, the receive descriptor count is an important parameter to avoid packet loss. For example if the driver has been configured with receive 16 descriptors and 17 packets arrives so quickly that the driver or stack hasn't had the same time to process the incoming Ethernet packet the hardware must drop the 17:th packet. On the other hand if a particularly problem maximally has 5 packets out at the same time, this may be the case of say two concurrent TCP/IP connections, it may be enough with 8 descriptors to avoid packet loss, thus saving about 1536 bytes per descriptor. Too few transmit descriptors may decrease performance, but does not result in packet loss.

4.5.2 LAN91C111, 10/100 Ethernet

The LAN91C111 Ethernet driver can be found under "Devices Drivers" in the VxWorks kernel configuration utility.

The PIO channel number is configurable via LAN91C111, it is important for the interrupt generation.

4.5.3 GRSPW, SpaceWire

The GRSPW SpaceWire driver can be found under "Devices Drivers" in the VxWorks kernel configuration utility.

The driver has four configuration parameters, making it possible to lower the memory resources needed by the driver. The maximal packet sizes limit per packet resources and the descriptor count the number of packets. The total size required by the driver can by calculated as below.

```
(TX_PKT_SIZE*TX_DESC_COUNT)+(RX_PKT_SIZE*RX_DESC_COUNT)
```

4.5.4 GRCAN, DMA CAN 2.0b

The GRCAN CAN 2.0b DMA driver can be found under "Devices Drivers" in the VxWorks kernel configuration utility.

The GRCAN driver uses two RAM areas that the CAN controller read or write messages to

using DMA. Depending on the system configuration the location of the RAM area may be important to avoid CAN message losses. The driver provides the GRCAN_RXBUF_ADDR and GRCAN_TXBUF_ADDR parameter to make it possible to manually set up the address of this area, setting zero will default the driver to use the dynamic memory allocation utilities (*memalign*) in the kernel to allocate the needed buffer. Note that the address has alignment requirements as described in the GRCAN hardware manual.

The size of the transmit and receive buffers can be controlled with the GRCAN_RX_SIZE and GRCAN_TX_SIZE parameters. The sizes must be a multiple of 16, this is because each CAN message is represented by 16 bytes by the GRCAN controller.

4.5.5 OCCAN, non-DMA CAN 2.0b

The OCCAN CAN 2.0b driver can be found under "Devices Drivers" in the VxWorks kernel configuration utility.

The OCCAN driver reads and writes CAN messages to an internal buffer in the OCCAN controller. To avoid CAN message losses the driver must take an interrupt for every received message to make room for more messages. Depending on the FIFO deep and the CAN baud rate the interrupt must not be delayed too long to guarantee no losses. To prevent messages losses from occurring the driver uses an internal software FIFO, and for performance reasons a software FIFO for transmissions . The lengths of the software FIFOs can be controlled with the OCCAN_RXFIFO_LEN and OCCAN_TXFIFO_LEN parameters.

4.5.6 GRUSBHC, Host USB 1.1, 2.0

The Host USB 1.1 and 2.0 controller can be enabled by adding the UHCI (1.1) and/or EHCI (2.0) USB stack, the stack can be found under "Hardware/buses/USB Host" in the VxWorks kernel configuration utility.

Additional USB class drivers can be added from "Hardware/peripherals/USB Devices". Adding support for USB Flash memories, for example, for USB 2.0 can be done by including INCLUDE_EHCI, INCLUDE_EHCI_INIT, INCLUDE_USB_MS_BULKONLY, INCLUDE_USB_MS_BULKONLY_INIT, FOLDER_DOSFS2, INCLUDE_FS_MONITOR. The USB flash disk partitions will appear as block devices under /bd0, /bd0:2, /bd0:3 and so on.

4.5.7 I2CMST, I2C Master

The I2C Master driver can be found under "Devices Drivers" in the VxWorks kernel configuration utility. A small I2C-EEPROM library using the I2C driver is available under "Device Drivers" as well.

4.5.8 APBUART

The APBUART driver is used by the standard terminal driver, it can be found under "Hardware/peripherals/serial/SIO" in the VxWorks kernel configuration utility.

The first NUM_TTY APBUART cores are used by the driver. The CONSOLE_TTY parameter selects what APBUART core shall be used as system console. The CONSOLE_BAUD_RATE selects the default and starting baud rate of the core.

For LEON2 systems there are two UARTs available and for LEON3 systems it depends on the system design.

4.5.9 PCI

The PCI support available for SPARC/LEON BSPs in VxWorks is summarized in the table below. The PCI Host support has been implemented using the vxBus PCI driver model. Auto configuration is default.

PCI Driver	Non-MMU	MMU	Component
AT697 PCI	Yes	No	INCLUDE_AT697_PCI
GRLIB GRPCI	Yes	Yes	INCLUDE_GRLIB_GRPCI_PCI
Actel CorePCIF - PCIF	Yes	Yes	INCLUDE_GRLIB_PCIF_PCI

Table 4.1: SPARC/LEON PCI support

The PCI support is included by adding the INCLUDE_PCI component from the workbench kernel configuration "Bus Support" section under hardware. One must also select one of the three Host PCI drivers available. The PCI address space can be configured from *config.h* or from the workbench kernel configuration utility INCLUDE_PCI_PARAMS component. VxWorks separates the PCI addresses by type, I/O space or memory space or non-prefetchable memory space. How the address space should be set up differ between systems depending on the needs of the PCI boards connected to the bus. The regions may not overlap.

The AMBA address space used to access the PCI address space is mapped 1:1.

The default is to map the SRAM or SDRAM of the CPU board to PCI address space on a 1:1 basis. This is to make the main memory available for cores on the PCI bus that supports DMA transactions.

On MMU systems the AMBA address window used to access PCI space has the same virtual address as the physical, thus virtual addresses is mapped 1:1 to physical addresses. This removes the need for device drivers to map their hardware to virtual space since all of PCI space is already mapped by the AMBA routines during startup.

Interrupts are either shared where multiple PCI interrupts (INT#A..D) share the same system interrupt, or non-shared where every used interrupt is assigned one unique system IRQ. The non-shared systems often use GPIO interrupt pins to connect one PCI interrupt to one system interrupt.

4.5.9.1 AT697 PCI

The LEON2 AT697 PCI has a fixed AMBA address window 0xA0000000-0xF0000000 translated into PCI addresses without address modifications. The address regions and sizes available to VxWorks must be configured using the INCLUDE_PCI_PARAMS component described above.

Interrupt are generated using the GPIO interface which is connected to one of the I/O interrupt sources. The GPIO number used can be changed by setting PCI_SHARED_IRQ_PIO, the system interrupt connected to the GPIO interrupt can be changed by setting PCI_SHARED_IRQ.

4.5.9.2 GRLIB GRPCI

The GRPCI core has a configurable AMBA address window to access PCI space. The address range is read out by the driver during initialization and mapped 1:1 to PCI space. This address space is also mapped into the virtual address space for MMU BSPs.

If the GRPCI shared interrupt service is used the GRPCI is responsible to route PCI interrupt to

a system interrupt, if this is the case the PCI_INT#_IRQ parameters will be ignored and the IRQ number will be taken from the AMBA Plug&Play GRPCI IRQ number. This is the default behaviour and GRPCI_VXBUS_GPIO_INTERRUPT must be set to FALSE.

However, if the GRPCI is not connected to the PCI interrupts, but GPIO is used to connect system interrupts with PCI interrupts set the PCI_INT#_IRQ to match the hardware design. The PCI_INT[N]_IRQ number represents the system IRQ the PCI_INT[N]# is connected to, all the system IRQs given will be enabled and configured as low level sensitive. Unconnected PCI IRQs are set to 0xff.

4.5.9.3 GRLIB PCIF

The GRLIB PCIF core is an AMBA wrapper for Actel CorePCIF. The interface the PCIF wrapper is similar to the GRPCI interface. The PCIF does not support changing byte twisting and PCI bridges. See GRPCI section above for PCIF guidelines.

4.5.10 1553

The three 1553 functions Bus Controller (BC), Remote Terminal (RT) and Monitor Terminal (MT) are all supported by the GRLIB B1553BRM driver for VxWorks SPARC/LEON BSPs. See the GRLIB IP Core User's manual for information about the hardware.

The driver operate in either BC, RT or MT mode. The mode is selected during runtime using the standard *ioctl* interface.

The driver can structure the DMA memory area in two different ways either using 128Kbyte or 16Kbyte. The memory structuring method must be selected during compilation time. The memory structuring is highly application specific and more ways can added by editing the driver. The memory structuring method can be selected by changing the component parameter B155BRM_MEM_LAYOUT, see below.

DMA memory region location can be manipulated using the component parameter B1553BRM_DMA_MEM_AREA, see below. The driver can be configured to use dynamical memory allocation methods, *mem_align*, to allocate the DMA memory needed by setting the parameter to -1. If the parameter is not set to -1 the driver will assume the parameter value is the location of a preallocated DMA area of the size determined by B1553BRM_MEM_LAYOUT. Note that the hardware requires the DMA memory area to be aligned on a 128Kbyte boundary.

The B1553BRM driver is added by including the DRV_GRLIB_B155BRM component, it depends on the vxBus layer. The configuration options available is described in the table below.

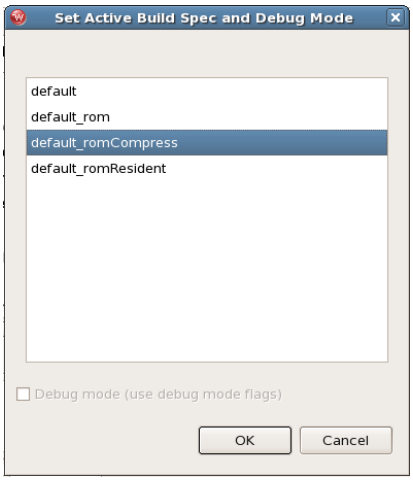
Name	Description
CLK_DIV	Clock division factor, not implemented on all hardware designs
CLK_SEL	Selects input clock, not implemented on all hardware designs
CLKFREQ_MHZ	Actel CORE1553BRM CLKFREQ option, 12, 16, 20 or 24 MHz
MEM_LAYOUT	Select memory structuring method
DMA_MEM_AREA	Configures how the DMA memory region is allocated by the driver

Table 4.2: B1553BRM parameters

5 BUILDING PROJECT

5.1 Build rule

By using the Workbench one can set the build rule and the Workbench remembers the setting until the project is built. Choose build rule by pressing right mouse button on the project icon and selecting 'Build Options/Set Active Build Spec...'. A dialogue presents all possible targets for the project:



- Default – image run from directly from RAM, no boot loader included. Typically used during debugging using grmon.
- Default_rom – image with boot loader not compressed. Programmed into flash. Boot loader copy kernel to RAM and starts execution from there.
- Default_romCompress – image with boot loader able to decompress kernel. Programmed into flash. Bootloader decompress kernel into RAM and jump to start of kernel thereafter.
- Default_romResident – image intended to run from flash memory. Programmed into flash memory. Kernel is not copied to RAM. Used where RAM space is an issue.

See Boot loader configuration for boot loader options.

The same targets apply for building projects with the command line tools. The build rule is passed as an environment variable during project building. The variable's name is ROM. If ROM is left unset the default target will be built, that is the RAM image. However, setting it to rom, romCompress or romResident selects one of the build rules described above. See Building the project for examples.

5.2 Building the project

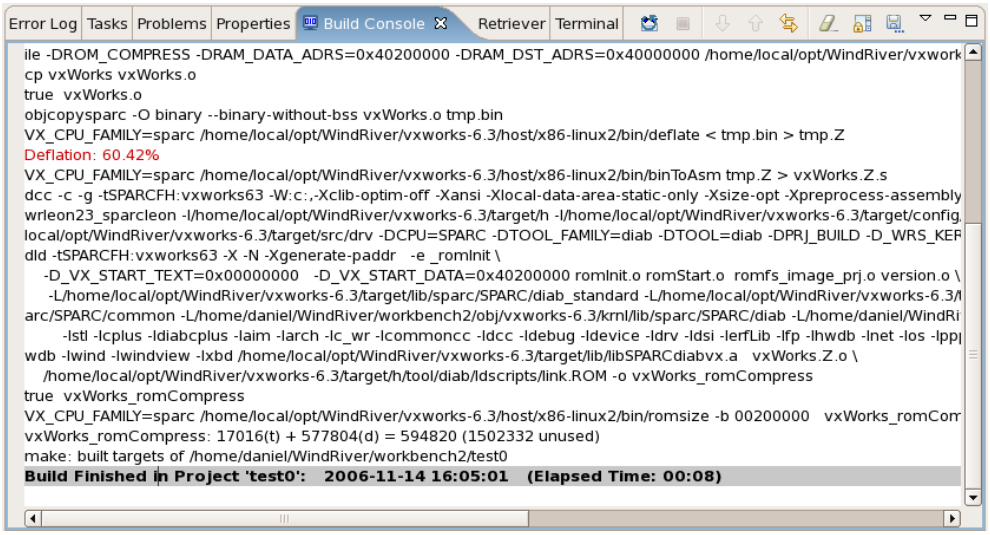
Compiling an image project with the command line tools is done by typing:

```
$ cd /path/to/distdir
$ NAME=prj make vxworks-usr-compile
```

Building an compressed image that will be burned to flash memory and run from RAM can be done by adding the ROM variable set to romCompress:

```
$ cd /path/to/distdir
$ NAME=prj ROM=romCompress make vxworks-usr-compile
```

One can also use the Workbench to build the project. Press Build and Build Project from the main menu while keeping the project highlighted in the project navigator:


Error Log
Tasks
Problems
Properties
Build Console
Retriever
Terminal
ile -DROM_COMPRESS -DRAM_DATA_ADRS=0x40200000 -DRAM_DST_ADRS=0x40000000 /home/local/opt/WindRiver/vxwork
cp vxWorks vxWorks.o
true vxWorks.o
objcopysparc -O binary --binary-without-bss vxWorks.o tmp.bin
VX_CPU_FAMILY=sparc /home/local/opt/WindRiver/vxworks-6.3/host/x86-linux2/bin/deflate < tmp.bin > tmp.Z
Deflation: 60.42%
VX_CPU_FAMILY=sparc /home/local/opt/WindRiver/vxworks-6.3/host/x86-linux2/bin/binToAsm tmp.Z > vxWorks.Z.s
dcc -c -g -tSPARCFH:vxworks63 -W:c:,-Xclib-optim-off -Xansi -Xlocal-data-area-static-only -Xsize-opt -Xpreprocess-assembly
wrleon23_sparcleon -I/home/local/opt/WindRiver/vxworks-6.3/target/h -I/home/local/opt/WindRiver/vxworks-6.3/target/config
local/opt/WindRiver/vxworks-6.3/target/src/drv -DCPU=SPARC -DTOOL_FAMILY=diab -DTOOL=diab -DPRJ_BUILD -D_WRS_KER
dld -tSPARCFH:vxworks63 -X -N -Xgenerate-paddr -e _romInit \
-D_VX_START_TEXT=0x00000000 -D_VX_START_DATA=0x40200000 romInit.o romStart.o romfs_image_prj.o version.o \
-L/home/local/opt/WindRiver/vxworks-6.3/target/lib/sparc/SPARC/diab_standard -L/home/local/opt/WindRiver/vxworks-6.3/
arc/SPARC/common -L/home/daniel/WindRiver/workbench2/obj/vxworks-6.3/krnl/lib/sparc/SPARC/diab -L/home/daniel/WindRi
-lstl -lcplus -ldiabcplus -laim -larch -lc_wr -lcommoncc -ldcc -ldebug -ldevice -ldrv -ldsi -lerfLib -lfp -lhwdb -lnet -los -lpp
wdb -lwind -lwindview -lxbd /home/local/opt/WindRiver/vxworks-6.3/target/lib/libSPARCdiabvx.a vxWorks.Z.o \
/home/local/opt/WindRiver/vxworks-6.3/target/h/tool/diab/ldscripts/link.ROM -o vxWorks_romCompress
true vxWorks_romCompress
VX_CPU_FAMILY=sparc /home/local/opt/WindRiver/vxworks-6.3/host/x86-linux2/bin/romsize -b 00200000 vxWorks_romCom
vxWorks_romCompress: 17016(t) + 577804(d) = 594820 (1502332 unused)
make: built targets of /home/daniel/WindRiver/workbench2/test0
Build Finished in Project 'test0': 2006-11-14 16:05:01 (Elapsed Time: 00:08)

The executable is located in the project directory under the chosen build rule, for example default_romCompress/vxWorks_romCompress.

5.3 Running image on hardware using GRMON

For flash targets using DSU via USB:

```
$ grmon -nb -usb
...
grlib> flash unlock all
grlib> flash erase 0 0xf0000
grlib> flash load prj/default_romCompress/vxWorks_romCompress
grlib> go 0
```

Running image from RAM during debugging:

```
$ grmon -nb -usb
...
grlib> load prj/default_romCompress/vxWorks_romCompress
grlib> run
```

6 BOOTROM

This section describes the main steps in creating a VxWorks bootrom using the Workbench GUI. The bootrom is typically used during the development process to download VxWorks kernels from a host on reset and power on. The host can distribute VxWorks kernels over the network using FTP, TFTP and TSFS host file system. See the VxWorks documentation for more information.

A bootrom can be created using either the command line tools or the Workbench GUI. The make targets *vxworks-bsp-compile* and *vxworks-bsp-res-compile* has been prepared for compiling the bootrom using the command line tools. It is also possible to create a bootrom using the command line tools manually as below,

```
$ make execute
sh-3.00$ cd vxworks-6.5/target/config/wrleon23_sparcleon/
sh-3.00$ make bootrom
```

The memory configuration in config.h must match the memory configuration in BSP/Makefile. In order to use the GNU toolchain to build the bootrom one must change BSP/ Makefile TOOL parameter to *gnu* instead of *diab*.

The default bootrom is configured with networking and GRETH driver, the driver can be changed by editing config.h.

Below is a description of how to create a bootrom using the Workbench.

6.1 Creating a bootrom project

A standard bootrom is created from the Workbench with the project creation guide, File -> New -> VxWorks Boot Loader Project. After giving the project a name select the non-MMU or the MMU BSP depending on the hardware configuration, build target to Resident and format to ELF.

A custom bootrom can also be created with the VxWorks Image Project guide by adding the INCLUDE_BOOT_APP component from the kernel configuration tool. This way the bootloader settings and bootrom features can be edited from within the configuration tool requiring no BSP files to be edited manually.

6.2 Configuring the bootrom

The bootrom is configured using the target/config/wrleon23[mmu]_sparcleon/config.h file for component selection and bootcfg_def.h for custom boot loader parameters. From config.h and Makefile it is possible to do the memory configuration needed, see the defines ROM_SIZE, ROM_BASE_ADRS, RAM_LOW_ADRS, RAM_HIGH_ADRS, LOCAL_MEM_LOCAL_ADRS, LOCAL_MEM_SIZE. The memory configuration must be the same for the bootloader and for VxWorks kernels loaded.

Depending on the boot method that component selection may vary. If networking is not needed or the LanChip driver is must be used it can be changed from within config.h. The bootrom network and boot settings can be changed by editing DEFAULT_BOOT_LINE. The boot line

argument definition can be found in the VxWorks documentation, below is a typical boot line booting the image VxWorks from host neptune (192.168.0.47) using the anonymous FTP service, the target itself has the IP-address 192.168.0.184 and name grxc3s1500,

```
greth0(0,0)neptune:vxWorks e=192.168.0.184 h=192.168.0.47
g=192.168.0.1 u=ftp pw=user f=0x04 tn=grxc3s1500
```

Hardware registers are initialized by the boot loader before the bootrom starts, in romInit.s values from bootloader.h and bootcfg_def.h are used to set up the registers. One must create a custom bootcfg_def.h in order for the boot loader to successfully initialize the system. The boot loader parameters are described in 4.1.

6.3 Building the bootrom

The bootrom can be built using different build targets bootrom, bootrom_uncmp, bootrom_res and bootrom_res_high. The default is bootrom. The bootrom target produces an image that will uncompress itself into RAM and run from RAM. The build target is selected when creating the project or by selecting Project -> Set Active Build Spec. The images bootrom and bootrom_uncmp images created during the build stage needs to be linked against ROM as below, however bootrom_res and bootrom_res_high does not require relinking.

```
sparc-rtems-objcopy \
--change-section-lma \
.text-(LOCAL_MEM_LOCAL_ADRS-ROM_BASE_ADRS) \
--change-section-lma \
.bss-(LOCAL_MEM_LOCAL_ADRS-ROM_BASE_ADRS)\
--change-section-lma \
.data-(LOCAL_MEM_LOCAL_ADRS-ROM_BASE_ADRS) \
--adjust-start -(LOCAL_MEM_LOCAL_ADRS-ROM_BASE_ADRS) \
bootrom bootrom.rom
```

Assuming RAM is at 0x40000000 and ROM at 0x00000000,

```
sparc-rtems-objcopy --change-section-lma .text-0x40003000
--change-section-lma .bss-0x40003000 --change-section-lma
.data-0x40003000 --adjust-start -0x40003000 bootrom
bootrom.rom
```

6.4 Installing and running the bootrom

The bootrom can be installed into FLASH using GRMON. The FLASH must be erased before written as below,

```
$ grmon -eth -ip 192.168.0.55
grmon> flash unlock all
grmon> flash erase 0 0x200000
grmon> flash load bootrom.rom
```

After a successful configuration the bootrom is booted after reset and power on, it can also be started from GRMON as follows,

```
grmon> run 0
```

The bootrom uses the serial terminal with the default settings as indicated by the table below. A terminal emulator can be started from within the workbench, Window -> Show View -> Terminal.

Setting	Value
Baud rate	38400
Data bits	8
Stop bits	1
Parity	None
Flow control	None

Table 6.1: Default terminal settings

7 RTP PROJECTS USING COMMAND LINE TOOLS

Real Time Process (RTP) projects can be made with the Workbench or the command line tools. This section only outlines the basic steps needed for RTP projects using the command line tools.

Real time processes run in different contexts, to make that possible MMU support is needed. Be sure to create projects based on the BSP supporting MMU (wrleon23mmu_sparcleon) when developing RTP projects.

Note that the GNU toolchain can not be used for RTP projects, the DIAB compiler must be used instead.

7.1 Overview

An RTP project usually consists of a file system where the RTP binaries are stored. The file system is part of a VxWorks image project. Example:

- Image project, build rule compressed ROM image
  A. Read only Filesystem accessed through /romfs, containing files:
     i. bin/rtp1.vxe – RTP project 1
     ii. bin/rtp2.vxe – RTP project 2
     iii. etc/rtp1.settings – Settings file for RTP project 1

It is similar to a normal PC computer. The Linux kernel is instructed to boot using a root file system in which one or more applications reside. One of the differences is that linux always starts /sbin/init whereas vxWorks kernel lets the programmer start processes from usrRtpAppInit.

7.2 Creating and Compiling a RTP Project

Create an RTP project by executing:

```
$ cd /path/to/distdir
$ NAME=testrtp1 make vxworks-rtp-create
```

The RTP project creator has created an Makefile and a simple C-code example that prints hello. See the project created in rtp/testrtp1.

Compile the project:

```
$ NAME=testrtp1 make vxworks-rtp-compile
```

7.3 Inserting an RTP project it into a ROMFS

After the RTP application binary is created one can copy the binary into the ROMFS directory manually or changing the compilation arguments so that it is automatically copied into place:

```
$ NAME=testrtp1 INSTALLTO=bin make vxworks-rtp-compile
```

The destination of the testttp1.vxe binary will be $ROMFS/bin/testrtp1.vxe. ROMFS is set up by editing the beginning of the Makefile.

7.4 Building image project with ROMFS and RTPs

A ROMFS is always compiled into the VxWorks kernel image when using the vxworks-usr-compile target. The directory pointed to by ROMFS will be seen as /romfs in the VxWorks command shell.

Create a MMU image project and compile it:

```
$ MMU=y NAME=test1 make vxworks-usr-create
$ NAME=test1 make vxworks-usr-compile
```

The resulting image (vxWorks) contains the read only file system with the RTP binary located in /romfs/bin.

7.5 Running real time processes on target hardware

On the system console of the target just enter the process name and it will be executed:

```
[vxWorks *]# ls -l /romfs
Listing Directory /romfs:
drwxr-xr-x  0 1015    100                56 Jun 29  1995 ./
drwxr-xr-x  0 1015    100                56 Jun 29  1995 ../
-rw-r--r--  0 1015    100                14 Feb 16  1972 README
drwxr-xr-x  0 1015    100                92 Feb 16  1972 bin/

[vxWorks *]# ls -la /romfs/bin
Listing Directory /romfs/bin:
drwxr-xr-x  0 1015    100                92 Feb 16  1972 ./
drwxr-xr-x  0 1015    100                56 Jun 29  1995 ../
-rwxr-xr-x  0 1015    100             86944 Feb 16  1972 testrtp1.vxe
-rwxr-xr-x  0 1015    100             94117 Feb 16  1972 helloworld.vxe

[vxWorks *]# /romfs/bin/testrtp1.vxe
Launching process '/romfs/bin/testrtp1.vxe' ...
Process '/romfs/bin/testrtp1.vxe' (process Id = 0x4034bfb8) launched.
hello this is rtp1

[vxWorks *]#
```

However, this approach is not always acceptable, for example in an production system one cannot ask the customers to login and enter a process name to start the application. Using rtpSpawn enables us to start the execution of an RTP process automatically at start up. rtpSpawn can be called from usrRtpAppInit in usrRtpAppInit.c in the image project (test1 in this example):

```
#include <vxWorks.h>
#include <rtpLib.h>
#include <taskLib.h>

void usrRtpAppInit (void){

  rtpSpawn("/romfs/bin/testrtp1.vxe",NULL,NULL,0,20000,0,VX_FP_TASK);

}
```

To create a usrRtpAppInit.c file the project, include the component INCLUDE_RTP by executing:

```
$ make execute
> vxprj component add usr/test1/test1.vpj INCLUDE_RTP
> vxprj component add usr/test1/test1.vpj \
INCLUDE_RTP_APPL_USER
> vxprj component add usr/test1/test1.vpj INCLUDE_RTP_SHOW
> exit
```

It is possible to include more RTP features in the same manner.

8 SUPPORT

For support, contact the Gaisler Research support team at support@gaisler.com